

A2 ISA. But, development of the new assembler may be the responsibility of a different team – often in a different geographic location, or worse, a different (third party) organization. This means that testing and debugging of the new ISA, or even the new ISS, must await completion of the new assembler. This makes it difficult for the assembler program to change quickly, much less allow it to change over a period of time as the extended ISA evolves. The developers of the extended ISA and new ISS must wait until the design and development of the new assembler is finished before using it to debug the extended ISA by compiling test programs, which may, in turn, necessitate changes in the assembler, and so on. This is a reiterative procedure that makes the overall task of changing processor/ISA designs a lengthy process. The current or old assembler is of no use in this development effort because it is incapable of properly interpreting and converting the new instructions.

Please replace three paragraphs beginning on page 4, line 13 to and including page 5, line 6, with the following:

A3 The present invention, as noted above, is a technique for employing an older assembler to produce executable object code from a source code containing old assembly language instructions, compatible with the older assembler, and added, new assembly language instructions not capable of being interpreted by the older assembler. The invention uses the data directive feature usually found in presently available assembler applications. Such data directive features are capable of taking an argument, usually in hexadecimal format, and inserting that argument in the object code unchanged.

Turning now to the figures, and for the moment Fig. 3, there is illustrated a diagrammatic representation of the method of the present invention as implemented on a processing system (not shown). As Fig. 3 shows, an original source file (File.asm) 40 contains old assembly instructions 40a (Old_inst_1 and Old_instr_2) and new assembly instructions 40b (movx.l @r1+r8, y1) that form a part of a new ISA. The original source file 40 is applied to a preprocessor (pp) software application 42. The preprocessor 42 operates to scan the source file 40, create a temporary source file 46, and write to the temporary source file 46, unchanged, the old assembly instructions 40a, 40a.

Each new instruction 40b encountered by the preprocessor 42 is checked for validity and, if found to be a valid instruction, converted to its object code equivalent. That object code

A3
equivalent is then also written to the temporary source code file 46 as the argument of a data directive 41, which usually takes the form of ".DATA [data]". Thus, as Fig. 3 illustrates, the new instruction 40b, "movx.l @r1+r8, y1", is converted to its object code equivalent, "12AB" (Hex), and inserted in the temporary source code file 46 as the argument of the data directive statement 41. Each data directive statement 41 will be placed in the instruction sequence of the temporary source code file 46 at the same location (relative to the other instructions 40) corresponding to where the new instruction 40b appeared in the original source code file 40.

Please replace the paragraph at appearing at page 5, lines 12-21, with the following:

A4
Although the old assembler is capable of converting the old instructions 40a directly to their object code equivalents, it would have been incapable of handling the new instructions 40b. However, when the old assembler 48 encounters a data directive in the source file, such as the data directive 41, it will use the argument of the data directive, the object code equivalent of the new instruction 40b, and insert that object code equivalent in the object file 50. What appears now in the object file 50 are the machine readable object code equivalents of both the old instructions 40a of the original source code 40 and, added as data by the data directives they were converted to, the object code equivalents of the new instructions 40b.

The replace the two paragraphs at page 5, lines 25-33 with the following:

A5
Turning now to Figs. 4A and 4B, the steps taken to assemble an assembly language program containing new and old instructions according to the invention is illustrated. Fig. 4A broadly shows the steps taken by a control script (NEWASM) 44, while Fig. 4B shows the principal steps taken by the preprocessor 42.

Turning first to Fig. 4A, when the control script 44 is invoked, at step 70, it will first call the preprocessor 42, passing to it two arguments: the identification of the source code file 40 and the name of the temporary output file (File.tmp) to be created. Control is then passed to the preprocessor 42, the main operative steps of which are outlined in Fig. 4B.

IN THE CLAIMS:

Please replace claims 1 and 2 with the following re-written claims 1 and 2: